# Patterns and Reuse in Document Engineering

*Keywords:* Document Engineering, Document Analysis, Data Analysis, Data Modeling, Patterns, Reuse, Methodology

Robert Glushko
University of California
Berkeley
California
US
glushko@sims.berkeley.edu
http://www.sims.berkeley.edu/~glushko

*Biography*

> Bob Glushko is a lecturer at the University of California at Berkeley in the School of Information Management and Systems and an Engineering Fellow at Commerce One. He founded or co-founded three companies, the last of which was Veo Systems in 1997, which pioneered the use of XML for electronic commerce before its 1999 acquisition by Commerce One.

Tim McGrath
University of Notre Dame, Australia
Fremantle
Western Australia
Australia
tmcgrath@portcomm.com.au

*Biography*

> Tim is recognised as a leader in the introduction of Internet technologies to the EDI and e-commerce marketplace in Australia. More recently, Tim led the Quality Review Team for the ebXML initiative, was a member of the ebXML Steering Committee and is a co-author of *Professional ebXML Foundations* from Wrox Press. Tim is currently the Chair of the Library Content subcommittee of the OASIS Universal Business Language (UBL).

---

## Abstract

---

"Document Engineering" is evolving as a new discipline for specifying, designing, and implementing the electronic documents that request or provide interfaces to business processes via Web-based services. The essence of Document Engineering is the analysis and design methods that yield formal models to describe the information these processes or services require. Good Document Engineering practice emphasizes the reuse of existing models wherever possible and requires that new models be described in ways that encourage their reuse by others.

Reusable patterns in Document Engineering are found at both the implementation level in the form of EDI and XML libraries and also at the conceptual level in terms of libraries of models that describe common business processes and the organization of activities between businesses. Re-using patterns at these more abstract levels facilitates interoperability between different technology implementations.

---

# Table of Contents

---



# Introducing "Document Engineering"

Businesses and individuals have long used documents to interact with each other and to describe their relationships, and there are hundreds of different types of documents in common use (for a colloquial list of document types, see Roget's Thesaurus **[RO 1962]** ; for more formal lists, see the United Nations Standard Messages (UNSMs) in EDIFACT **[UN Web]** or the XML schemas registered at xml.org **[XO Web]** ). While businesses always strive to differentiate themselves, their common goals for profitability and growth, common external forces like competition and regulation, and overlapping business relationships with common suppliers or customers drive them to do things in similar ways. In particular, the fundamental requirement that their documents must be mutually intelligible for a business relationship to be possible inevitably causes both the document models and the sequence of exchanges between businesses to follow regular patterns.

As companies increasingly move their existing activities to the Internet and experiment with new ways of doing business, it is even more beneficial to treat documents as interfaces **[GL 1999]** . Doing so presents a clean and stable relationship to business partners and customers that is insulated from changes in either the processes that create and consume the documents or in the technology with which these processes are implemented. Furthermore, when businesses exchange information using documents, their well-defined structure should enforce an interpretation or context in an unambiguous and efficient manner.

Many kinds of documents are essential to business. Some, like catalogs, brochures, and datasheets, assist

buyers in locating and selecting products and services. Others, like user guides, reference information, and maintenance manuals, are needed to make effective use of products and services after they are purchased. In its early days the Web was used primarily as a publishing or distribution medium for these kinds of non-transactional documents. Later, when the Web joined existing EDI technologies to conduct business, transactional documents like orders, invoices, and payment instructions evolved from printed forms to become important electronic document types.

But how do we know what specific information these documents must contain and how do we ensure that their recipients understand them? For the non-transactional types of documents, those that are traditionally called "publications," the analysis and modeling techniques used to answer these questions are usually described as "document analysis" (see, for example **[MA 1996]** , **[MU Web]** ). In contrast, transactional documents are optimized for use by business applications and differ in other substantial ways from traditional user-oriented publications. The methods used to answer these questions for transactional documents are often described as "data modeling" (see, for example **[CA 2001]** , **[HA 1996]** ). Neither analysis approach as currently described and taught is well suited for the other kind of document.

"Document Engineering" is evolving as a new discipline for specifying, designing, and implementing the electronic documents that request or provide interfaces to business processes via Web-based services **[GL 2002b]** , **[MC 2002]** . The essence of Document Engineering is the analysis and design methods that yield formal models to describe the information these processes or services require. The methods of Document Engineering are practical and effective for both transactional and non-transactional document types. The resulting models must be carefully designed to contain enough structure and semantics to convey meaning while not being so general (as are relational data models) that they allow too many interpretations. These models must also find a balance between the optimal document designs for a business's internal processes and the need for those documents to be understood by other businesses. This tension induces document designers to reuse existing models wherever possible and reinforces them if they describe any new models they create in ways that encourage their reuse by others.

A focus on the patterns in document models and in document exchanges doesn't preclude consideration of other kinds of patterns that businesses follow. For example, business activities can be organized by common function, product line, customer segment, geography, and so on, and each of these exhibits characteristic patterns in an organization chart. Likewise, how a business works can be described in terms of the architecture by which its software components are implemented and interconnected, with patterns for distinct styles of system communication or integration (N-tier, message oriented middleware, etc.). But we believe that the inherently static organizational perspective is less useful in discussions of web services and "virtual enterprises" created by building on and interconnecting the activities of businesses around the globe. Similarly, we believe that understanding the patterns in document models and document exchanges is a pre-requisite for making decisions about software architecture and not vice versa.

# Analysis Foundations for Document Engineering

## Document Analysis

Most document analysis is conducted with the goal of abstracting a logical model from existing instances of a single document type and then encoding the model in an SGML or XML schema. The optimal prescriptive model for the class of documents is that which best satisfies the requirements of current and prospective users for carrying out specific tasks with new instances. When the document types are "publications" of one sort or another, the new schema separates the description of a document's

structure and content from its presentational characteristics. These include the fonts, type sizes, and formatting attributes that are used to represent or highlight various structural or content distinctions. Once this separation is accomplished, one or more stylesheets can be used to assign formatting or rendering characteristics in a consistent manner to any valid instance.

The term "document analysis" is a misnomer because a good analyst usually interviews current and prospective users and considers the technology contexts of the document lifecycle. But the term sticks because documents often serve as proxies for the domain expertise that is the standard source of requirements information.

Because of the complex ways in which publications merge presentation with structure and content, the document analysis techniques developed by publishing experts emphasize the study of documents as artifacts that must each be studied very carefully. The more heterogeneous the instances of a given type, the more document analysis becomes descriptive rather than prescriptive, and the purpose of analysis is to identify the markup for "text encoding" that captures the specific and idiosyncratic character of each instance. This is especially the case when the document instance is so unique or important that we know it by name: *Magna Charta*, *Don Quixote*, *Declaration of Independence*, *Gettysburg Address*. While these could be analyzed as instances of document types like **Contract**, **Novel**, **FormalStatement**, and **CommemorativeSpeech**, respectively, what makes these document instances distinctive can't be captured in a formal model and can't easily be reused in new documents.

## Data Modeling

Data modeling has its roots in philosophy and linguistics but in its current incarnation as a methodology for designing information systems data modeling is primarily devoted to understanding and describing the logical structure of data objects that have various properties and associations with each other. The typical goal of data modeling is to define one or more categories or schemas that organize these properties and associations efficiently for creating, revising, or deleting data objects or for finding those with specific characteristics.

Data modeling shares the goal with document analysis of creating a formal description of a class of instances, but its methods apply best when there are a limitless number of essentially identical instances, often produced mechanically to represent some state of an activity or business process. Such documents or messages are extremely regular in their logical structures, have strongly typed content, and have minimal or arbitrary presentation features. Furthermore, a great deal of theory has evolved to guarantee that models are optimal with respect to their use in information systems.

## Contrasting Document Analysis and Data Modeling

One important way in which document analysis and data modeling techniques differ is in the nature of information reuse they facilitate. When applied to technical publications or similar document types, document analysis can identify reusable boilerplate content or reusable structures for things like tables, notes, lists, or phrase-level markup. In contrast, much greater reuse and finer granularity can typically be enabled by data analysis techniques because of the greater amount of structure and regularity exhibited by transactional documents. Furthermore, by their nature transactional documents often arise as transformationally related request-response pairs or as a result of correlated business activities, implying the need to model the overlapping content that is reused as it "flows" from one document type to another.

The rapid evolution in the past few years of the document-centric architecture for the Internet and web

services has also exposed weaknesses in traditional document analysis techniques. Some of the modeling and design methods for document analysis have been constrained by the well-known limitations of DTDs for encoding models. DTDs have weak datatyping and define most content as text. This reflects the DTD's heritage in publishing, but it becomes a severe limitation when the domain of document analysis is more data-intensive and a richer type repertoire is required. The limits of DTDs aren't limits of document analysis *per se*, of course, since the design of a document model is logically prior to its encoding in SGML or XML syntax. But to the extent that the tools most used by document analysts for developing and testing models evolved from publishing and rely on DTDs, the limits are real. No similar constraints on what can be modeled are embodied in the traditional tools of data modeling.

In addition, the paradigm shift in distributed computing introduced legions of software designers to XML document design, to which they looked for methods similar to those they used in the design of database schemas or object classes. They wanted to apply object-oriented techniques to define one element to be a template or archetype for another and to use concepts of extension and inheritance to reuse definitions in an efficient way across a set of related document types. Unfortunately, DTDs were not "programmer-friendly" in these ways and while more expressive schema languages for XML now exist, little of the new modeling capabilities they enable has been embraced by traditional document analysis practice.

# Unifying Document Analysis and Data Modeling in Document Engineering

The document analysis and data modeling perspectives come from different disciplines and use different tools, terminology and techniques. "Document" people and "data" people haven't known how to talk to each other and haven't often recognized the overlap in their goals, if not in their methods. Both offer valuable insight into designing effective documents but until now they have had little intersection. Document Engineering can unify these two perspectives by identifying and emphasizing what they have in common rather than highlighting their differences.

## A Document-centric Version of the Classical Analyze/Design/Refine Approach

Almost every book about systems design introduces some variation of an Analyze/Design/Refine methodology. According to this classical approach, the artifacts of the "real world" are analyzed and the results of this analysis are represented in a physical model that captures the characteristics of the artifacts as they exist in some context and expressed in a particular technology. Then the model is progressively refined into a more conceptual, logical model by identifying repeating or reoccurring structures, removing redundancies and technology constraints, and otherwise creating a more abstract, concise, and context-free representation of the essential characteristics. Finally, the refined model is implemented "in the real world" by expressing it in technology appropriate for the contexts in which it will be used.

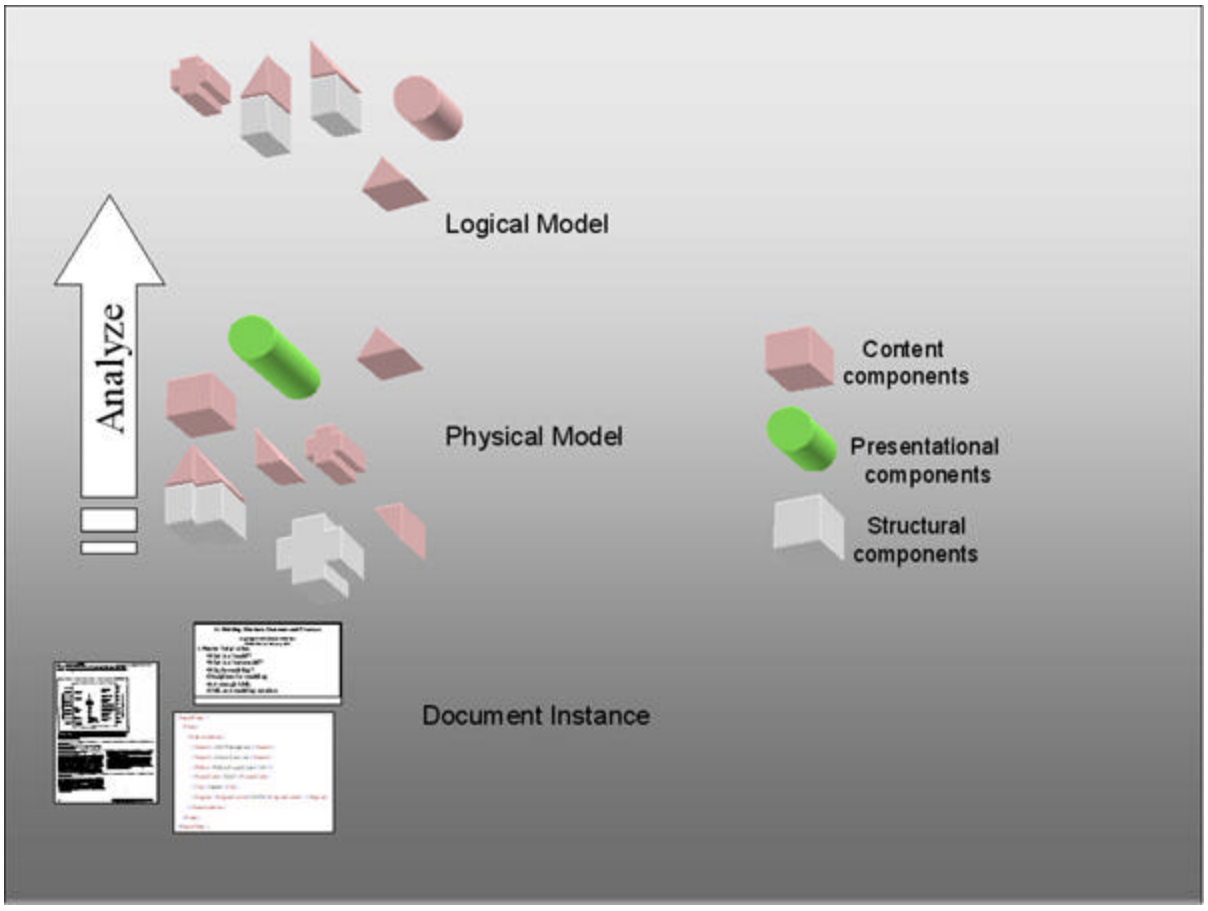
This classical approach is familiar to data modelers but can seem somewhat alien to document analysts. The traditional subject domain for data modelers consists of large numbers of identical instances, so the analysis activity isn't as document driven as it was when there are fewer and more heterogeneous instances to study. It is harder to "let go" of the artifacts and create logical models when the artifacts are more salient. In addition, document analysts, especially those who learned the skill when only DTDs were available to encode models, are more likely to think in terms of the modeling restrictions imposed by this syntax and thus are less inclined to spend significant time refining models at conceptual level.

What we call Document Engineering is at its core a "document-centric" version of the Analyze/Design/Refine methodology. We have introduced it to people familiar with either document analysis or data modeling (but not both) and it seems successful at bridging the two perspectives. We have also been encouraged by the emergence of a new generation of XML and modeling tools like XML Spy that share features with integrated programming environments. These provide more native support for data modeling, emphasize XML schemas rather than DTDs, and better integrate XML document models with database schemas and programming language class development. We believe that such tools will assist those with more traditional document analysis experience to rapidly acquire the complementary skills of data modeling and vice versa.

## Three Kinds of Components and Their Relationships

Documents contain three kinds of information components: content, structure, and presentation. Content components are always the most fundamental, but it is usually important to analyze the relationships of the other types of components as well. Structural components like chapters, sections, tables, headers and summaries, usually have some implicit semantic value because of their conventional use to reinforce a logical content hierarchy. A critical task in document analysis is determining the rules by which presentational information identifies or signals components of the other two types, because the visual design of printed or rendered instances can be highly complex.

**Fig. 1** illustrates how analysis is the process of identifying and separating these three types of components. It also suggests how the idiosyncratic characteristics of document instances need to be carefully analyzed in order to identify "good" logical components for potential reuse.



Analyzing Documents To Identify Component Types

In data modeling, because the structural organization of content is more regular and because the binding of presentational information to structure and content is less intrinsic for data-centric document types, analyzing structure and presentation is often seen as an afterthought. This contrast makes the vocabulary and methods of document analysis and data modeling seem more different than they actually are.

We have found that the presentational complexity and diversity of traditional publication types like encyclopedias, dictionaries, and reference manuals makes them good subject matter for teaching document analysis and design skills that can be transferred to transactional document types where the presentation cues are weaker. We have also emphasized the similarity between applying different presentation styles to document instances and generating different views or reports for collections of data. It is pedagogically helpful to discuss both in terms of "applying transformations" to the logical model and emphasizing the role of stylesheeting (e.g. XSLT) in both cases.

## Identifying "Good" Content Components and Component Assemblies

Each of the three component types has a different set of principles for achieving a quality design. We focus here on those or content components, and explain concepts and methods that applicable to a broad range of document types.

Content components can be identified at three levels **[CC Web]** :

1. "Atomic" components that hold individual pieces of information and that are typically represented by primitive data types (e.g., "string," "Boolean," "date") or datatypes readily derived from these.
2. "Aggregate" components that hold logically related groups of atomic components and sub-aggregates.
3. "Document" components that assemble atomic and aggregate components to form a self-contained logical unit of work; the clearest examples are transactional messages within a business process. The business process context provides the requirements for which documents are to be assembled.

The hardest level at which to identify good components is at the aggregate level. There is little doubt that we need some grouping of elements at the sub-document level both in our logical models and our schemas, but if we do this on an ad hoc and intuitive basis we might not identify the optimal patterns for re-use. For example, it might "sound right" to group **Name**, **Address** and **DateOfBirth** into an aggregate component of **Person**. But what is it about the relationships among these three components that makes them into a good aggregate?

The answer comes from conventional data modeling practice, which includes formal rules for designing logical structures and establishing what data analysts call functional dependencies in order to create modular and self-contained groups that lend themselves naturally to re-use. Much of what document analysts have done in the past, albeit informally, is applying similar principles to identify reusable components in logical models of documents. In Document Engineering we make this practice explicit so as to apply the same rigor to document schema design that we have customarily applied to data modeling.

### Functional Dependency and Normalization

Dependency means that if the value of one component changes when another component's value changes, then the former set is functionally dependent on the latter. For each **Person** we identify, there is

a different **Address** and **DateOfBirth** component because the values of each of these components functionally depend on the identity of the **Person** in question.

Data analysts use a formal technique for identifying and defining these dependencies, known as normalization **[DA 1981]** . Normalization is a series of analytic steps that:

1. Ensures that all data elements in a group are discrete, i.e., can only take a single value. This means we won't find lists of repeating values in any logical group.
2. Establishes the primary identifier of each logical group.
3. Aggregates groups of components that are fully dependent on each value of the primary identifier, i.e., for each instance of the set.
4. Ensures that all members apart from the primary identifier are independent of one another.

Normalization yields models that describe the network of relationships between logical groups of components in optimal ways that minimize redundancy and prevent inadvertent errors or information loss when components are added or deleted.

**Normalized Relational Models**

For example, if we analyze the business processes of a university we might identify two models for aggregate components called **Student** and **Course**. The standard business rules for universities allow a given **Student** to enroll in many **Courses** and for a given **Course** to be enrolled in by many **Students**. The principles of normalization would lead us to create a third aggregate component, which we might call **StudentCourse**, to represent the cross-reference between these two other components. A relational database organized in this way would allow us to retrieve the courses that any given student was taking (the student's **Transcript** or **CourseList**) or the students who were enrolled in any given course (the **CourseRoster**).

In another domain such as procurement, an **Order** may contain many **Products** (such as seen on a **PurchaseOrder** document) or a **Product** may be on many **Orders** (such as seen on a **SalesReport**). Normalization would introduce a **LineItem** component to reconcile these many-to-many relationships.

**Assembling Hierarchical Document Models**

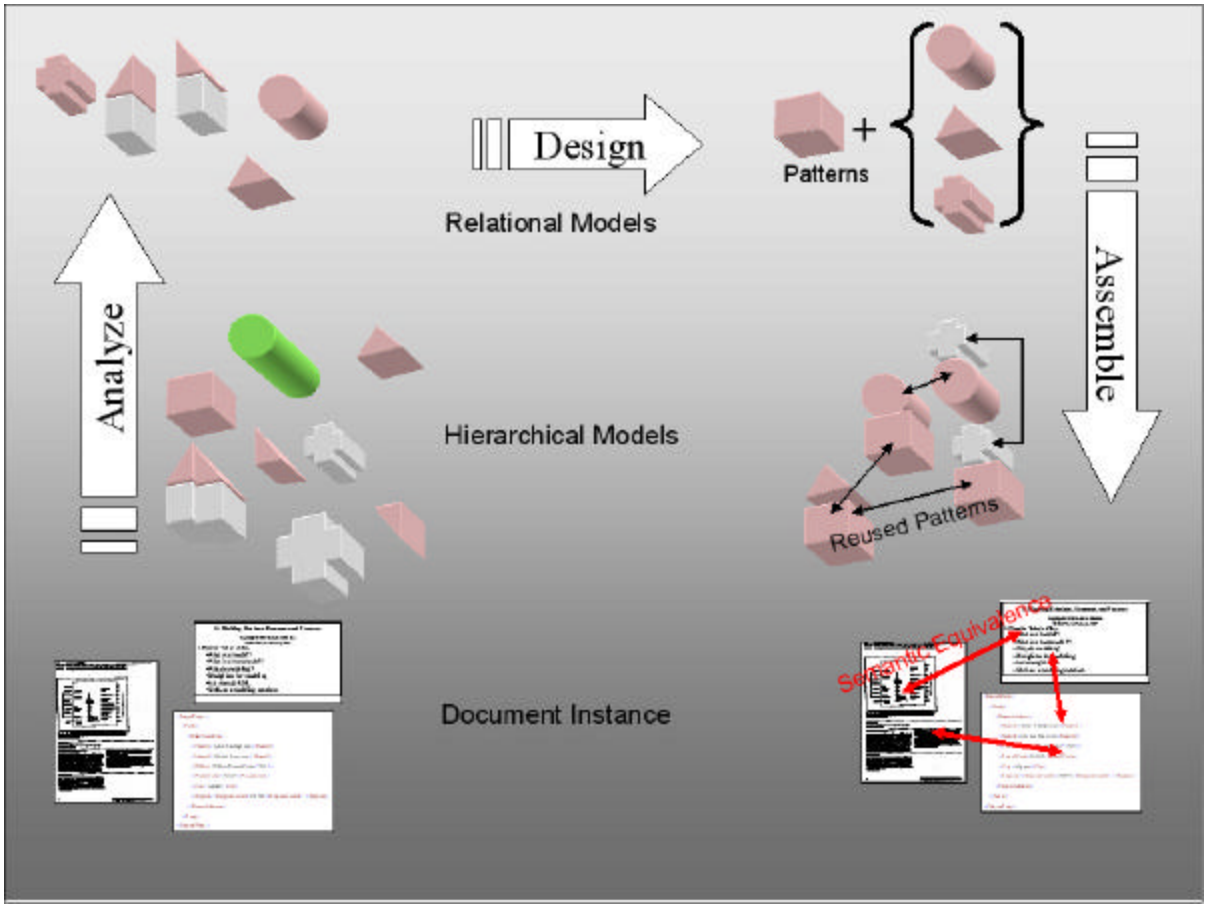
Two-way relationship patterns like these are common in relational data models and they provide great flexibility in the way we can maintain our information. They are an attempt to reflect the complex network or web of associations that exist in the real world.

However, when we want to exchange information with others, this flexibility amounts to ambiguity. We do not want to show all the relationships among the information components, only those that are relevant to the business context we are in. This context-specificity is best achieved by creating (or assembling) a hierarchical view out of the relational representation. Hierarchical views introduce container structures to impose a particular interpretation on the information we want to exchange.

Of course, we can assemble several alternate hierarchical views of the same relational model, as we saw with **Student** and **Course** (**Transcript** vs **CourseRoster**) and **Order** and **Product** (**PurchaseOrder** vs **SalesReport**). If we need to create a schema for a **PurchaseOrder** document type we would start at **Order** and list all **LineItems** and their associated **Products**. If we wanted a **SalesReport** document type schema we would start at **Product** and list all **LineItems** and their associated **Order**. The contrasting document schemas reuse the same components but assemble them in two different container structures,

one the inverse of the other.

In this way, the hierarchical view both enforces integrity rules and prevents ambiguity in the meaning of the data. What we are saying when we assemble a hierarchical view is "we want to emphasize one context in which you are to understand the data this way." This additional step of assembling relational components into hierarchical documents to establish context is what makes Document Engineering a distinctive methodology and not just a style of data modeling. **Fig. 2** illustrates the roles of analysis, model refinement, and assembly in the methodology of Document Engineering.




The Methodology of Document Engineering

Once it is assembled by following a one-way path through the relational model, the hierarchical model can be directly implemented as an XML schema. This document schema need not show all components and their relationships as described in the relational model, only the ones pertinent to our business context. Put another way, what this means is that logical components are patterns that can be re-used by assembling them into document schemas based on the context of their use.

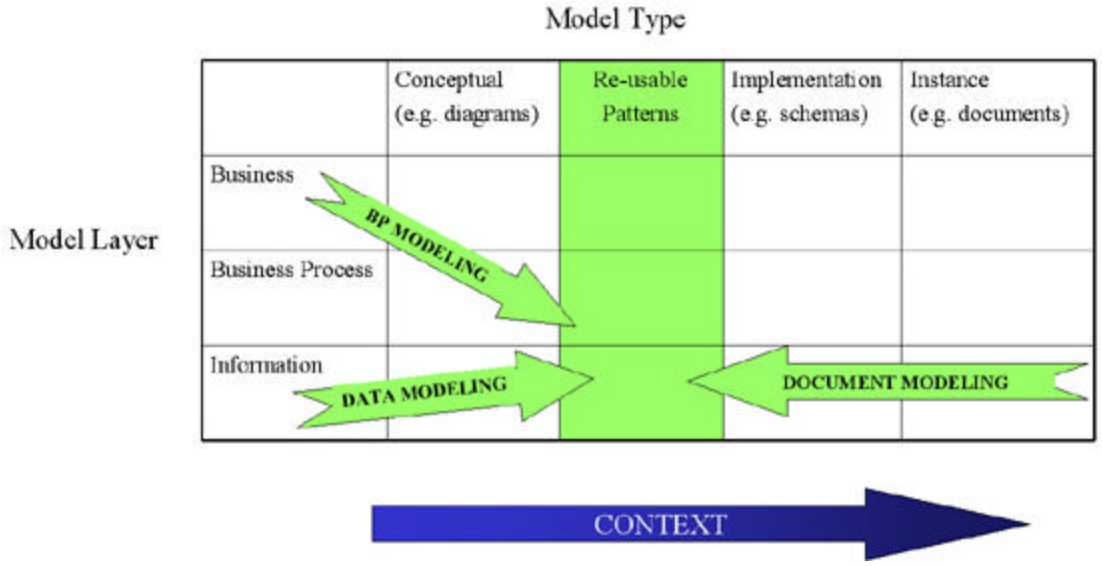
## Organizing Patterns to Facilitate Reuse

Reuse of patterns has the immediate benefit of reduced design and maintenance effort, encouraging and reinforcing consistency and standardization. Effective analysis enables us to recognize when a pattern can be reused, when a new pattern should be created, and what contexts distinguish one pattern from another.

Patterns useful in Document Engineering are found at both the implementation level in the form of EDI and XML libraries and also at the conceptual level in terms of libraries of models that describe common business processes and the organization of activities between businesses. Re-using patterns at these more abstract levels facilitates interoperability between different technology implementations.

**The Reuse Matrix**

Models and patterns vary in both their level of abstraction and in the granularity with which they view business, so it is helpful to organize all of them in a single framework, which we call the "Reuse Matrix" ( **Fig. 3** ).




The Reuse Matrix

We depict four levels of abstraction or model types, varying from highly abstracted logical or conceptual models to specific instances of documents needed to implement a model. The center of this dimension is where the "sweet spot" of patterns is found; patterns here have enough abstractness to be reusable, but enough concreteness to be prescriptive. For example, IBM's "Patterns for e-business" **[AD 2001]** falls squarely in the cell for conceptual business and business process patterns and Silverson's "Data Model Resource Book" **[SI 2001]** is a set of patterns for conceptual information components

Against the levels of abstraction we also have the depth at which we describe the patterns of document exchanges that are required by each model. These descriptions range from patterns of document exchange viewed at the "business to business" level, such as "vendor managed inventory" or "build to order," to patterns of exchange from the perspective of a single business (such as a RosettaNet Partner Interface Process **[RN Web]** ), to the most granular perspective that shows the re-use of components such as the common Address structure in XML schema libraries like xCBL **[XC Web]** .

The common goals but different approaches of document analysis and data modeling can be depicted in the Reuse Matrix, as can the relationship of these two approaches to information analysis to business process analysis. We suggest that document analysis typically starts from the lower right (where instances of data components in document artifacts can be found), while data modeling generally starts from the lower left (focusing on logical models of objects and associations). Business Process analysis typically starts from the upper left (abstract views of high level business models). Note that all three of

these approaches need to reach the same place in the Reuse Matrix, the point at which most reusable patterns exist. In practical terms this convergence depicts the key principles of Document Engineering that models for documents and business processes need to be developed at the same time, with the same care, and to compatible levels of detail.

The concept of context, which we described when we explained the assembly of document models, can also be represented in the Reuse Matrix as the increased specificity of models as we move to the right in any row. For example, in the top row what starts in the upper left corner as the abstract business model of "supply chain" becomes a more prescriptive pattern for "vendor managed inventory" in the context of retailing for packaged consumer goods as we move to the right (see **[SC Web]** ). In the bottom row at the information model level, the common **Address** structure at the conceptual level would, in the context of an **Order** XML Schema, be implemented as an **AddressType**. This is a pattern that may be re-used as **ShipToAddress** or **BillToAddress**, for example. At the further right hand side, we would find specific instances of this implementation. These would be in the context of specific addresses to be shipped or billed to, as they appeared on the **Orders**.

# Applying the Methods of Document Engineering

We first tested this new methodology of Document Engineering in a graduate course titled "Document Engineering for E-business" at the University of California, Berkeley, in the Spring of 2002 taught by the first author **[GL 2002a]** . Students were taught document analysis and data modeling skills, introduced to business-to-business, business process, and document patterns, and applied the principles of Document Engineering in the design of an "electronic university."

An early assignment required students to apply the patterns of "build to order" (used so successfully by Dell Computer as a manufacturer of personal computers **[KR 1996]** ) and "marketplace" to the operation of the university, which of course has more than once been called a "degree factory." Some students at first resisted the extent of commercialization implied by viewing students as product consumers and their professors as suppliers of those products. Later, though, after applying the commercial patterns to a new domain, most students appreciated that a "Dell-iversity" might offer them more course choices and customized majors, though perhaps only by eliminating tenure and research activity, which can be viewed as "friction" in the supply chain.

Another assignment required students to develop a set of related models for three document types along with a library of components that could be used by all of them. Some students were assigned highly regular transactional document types like **CourseRosters** and **Transcripts** while others were given document types like **GraduationRequirements** that had less transactional character. Students were able to reuse the same aggregate components like **Student** and **Course** by assembling them in different hierarchies. As might be expected, the modeling tasks were more difficult for those whose document sets contained the non-transactional documents because the document instances for those types were more heterogeneous and yielded less reusable aggregate components.

In a third assignment students modeled the process of selecting and enrolling in courses to the point in the semester where they receive a bill from the university. They were required to use PIPs from the RosettaNet directory **[RN Web]** as patterns for the business transactions needed in their model. Despite the fact that these PIPs were designed to standardize the business processes in information technology and semiconductor product supply chains, students were quite successful in reusing them in the university "supply chain." Students used PIPs for querying product information, requesting availability information, creating and managing purchase orders until the buyer and supplier have an agreement, and sending invoices. That's the pattern of procurement whether one is buying steel, paper clips, or

university courses.

We are encouraged by the success we've had in creating a systematic methodology of Document Engineering. We've been able to bridge the traditional gap between document analysis and data modeling with methods that are practical and effective for both transactional and non-transactional document types. Our Reuse Matrix appears to be a useful framework for organizing models and patterns of different levels of abstraction and in the granularity with which they view business.

We are now applying Document Engineering principles and methods to problems of larger scale, including the effort to develop a Universal Business Language (**[UB Web]** , **[MC 2002]** ) and the E-Berkeley initiative **[EB Web]** at the University of California, Berkeley.

# Bibliography

**[AD 2001]**
Adams, J., Koushik, S., Vasudera, G., and Galambos, G. *IBM Patterns for e-business: A Strategy for Reuse*. IBM Press, 2001.

**[CA 2001]**
Carlis, J., and Maguire, J. *Mastering Data Modeling*. Addison Wesley, 2001.

**[CC Web]**
Context and reusability of core components. 10 May 2001. http//www.ebxml.org/specs/ebCNTXT.pdf

**[DA 1981]**
Date, C.J. *An Introduction to Database Systems 3rd Edition*. Addison-Wesley, 1981

**[EB Web]**
http://eberkeley.berkeley.edu/

**[GL 1999]**
Glushko, R., Tenenbaum, J. and Meltzer, B. An XML framework for agent-based e-commerce. *Communications of the ACM*, (March 1999, 42(3), 106-114).

**[GL 2002a]**
Glushko, R. *Document Engineering for e-Business*, University of California, Berkeley, Spring 2002. http://www.sims.berkeley.edu/academics/courses/is290-4/s02/

**[GL 2002b]**
Glushko, R., and McGrath, T. Document Engineering for e-Business. *ACM Symposium on Document Engineering*, 2002.

**[HA 1996]**
Hay, D. *Data Model Patterns: Conventions of Thought*. Dorset House Publishing, 1996.

**[KR 1996]**
Kraemer, K., Dedrick, J., and Yamahiro, S. Refining and Extending the Business Model with Information Technology: Dell Computer Corporation. *The Information Society*, 16, 5-21.

**[MA 1996]**
Maler, E, and Andaloussi, J. *Developing SGML DTDs: From Text to Model to Markup*. Prentice Hall, 1996.

**[MC 2002]**
McGrath, T., and Glushko, R. Universal Business Language (UBL) Position Paper: Library Content Methodology. 26 June 2002. http://oasis-open.org/committees/ubl/lcsc/doc/position-mcgrath-methodology-01.doc

**[MU Web]**
Mulberry Technologies, Inc. *Data Modeling and XML Vocabulary Development.* http://www.mulberrytech.com/papers/xmlvocab.pdf

**[RN Web]**
RosettaNet PIP Directory
http://www.rosettanet.org/rosettanet/Rooms/DisplayPages/LayoutInitial?Container=com.webridge.entity.Entity%5BOID%5B9A6EEA233C5CD411843C00C04F689339%5D%5D

**[RO 1962]**
*The Original Roget's Thesaurus of English Words and Phrases.* Dell Publishing, 1962. (See especially Section 520, "Publication" and Section 589 "Book")

**[SC Web]**
Supply-Chain Council. *Supply-Chain Operations Reference-model: Overview of SCOR Version 5.0.* 2001. http://www.supply-chain.org/slides/SCOR5.0OverviewBooklet.pdf

**[SI 2001]**
Silverston, L. *The Data Model Resource Book Volumes I and II.* John Wiley and Sons, 2001

**[UB Web]**
UBL: The Next Step for Global E-Commerce. http://www.oasis-open.org/committees/ubl/msc/200112/ubl.pdf

**[UN Web]**
UN/EDIFACT Standards http://www.unece.org/trade/untdid/directory.htm

**[XC Web]**
XML Common Business Library. http://www.xcbl.org/about.html

**[XO Web]**
XML.ORG Registry. http://www.xml.org/xml/registry.jsp